Tutorial: Creating a Customer Support Bot Using ChatGPT

In this tutorial, we'll walk through how to create a basic customer support bot using OpenAI's GPT-3.5/4 (ChatGPT). We will focus on building a Python-based chatbot that can handle user queries and provide customer support by using the GPT API. We'll also cover how to deploy this bot on a web platform using Flask, a Python web framework.

Prerequisites

1. **Python 3.7 or higher** installed.
2. **OpenAI API Key** (you can sign up for an API key at OpenAI).
3. Basic understanding of Python, APIs, and Flask.

Step 1: Install Required Libraries

Before we start coding, you need to install the necessary Python packages. Open your terminal and install the following libraries:

```
pip install openai flask
```

- `openai`: This library allows us to interact with OpenAI's GPT API.
- `flask`: Flask will be used to create a simple web interface for our bot.

Step 2: Set Up OpenAI API in Python

First, we need to set up the Python environment and connect it to OpenAI's API. Here's a simple script that sends a user query to GPT-3.5/4 and gets a response.

python

```
import openai

# Set up your OpenAI API key
openai.api_key = 'your-api-key-here'

def get_bot_response(user_query):
    # Call GPT-3.5/4 API to generate a response
    response = openai.Completion.create(
        engine="gpt-4",  # Can use "gpt-3.5-turbo" or "gpt-4"
        prompt=f"User: {user_query}\nBot:",
        max_tokens=150,  # Limit the response length
        n=1,
```

```
        stop=None,
        temperature=0.7,  # Control creativity of the bot
    )

    # Extract the bot's response from the API response
    bot_response = response.choices[0].text.strip()
    return bot_response
```

- Replace `'your-api-key-here'` with your actual OpenAI API key.
- The `max_tokens` parameter controls how long the response will be.
- `temperature` affects randomness in responses (lower values mean more focused responses).

Step 3: Create a Simple Flask Web App

Next, we'll create a web interface using Flask to make it easier for users to interact with the bot. This part of the code sets up a basic form where users can type their queries, and the bot responds.

Here's the Flask app:

python

```
from flask import Flask, render_template, request
import openai

# Initialize Flask app
app = Flask(__name__)

# Set your OpenAI API key here
openai.api_key = 'your-api-key-here'

# Define a function that interacts with GPT-3.5/4 API
def get_bot_response(user_query):
    response = openai.Completion.create(
        engine="gpt-4",  # You can also use "gpt-3.5-turbo"
        prompt=f"User: {user_query}\nBot:",
        max_tokens=150,
        n=1,
        stop=None,
```

```
        temperature=0.7,
    )
    bot_response = response.choices[0].text.strip()
    return bot_response

# Define the main route for the bot interface
@app.route("/", methods=["GET", "POST"])
def chatbot():
    if request.method == "POST":
        user_query = request.form["user_query"]
        bot_response = get_bot_response(user_query)
        return render_template("index.html", user_query=user_query,
bot_response=bot_response)
    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)
```

- The `chatbot` function handles both GET and POST requests.
- When the user submits a query, it sends the query to the GPT model using `get_bot_response()` and displays the bot's response.

Step 4: Create an HTML Template

Next, you need an HTML template (`index.html`) to render the bot's web interface. Create a folder called `templates` in the same directory as your Python script, and within it, create an `index.html` file with the following content:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Customer Support Bot</title>
    <style>
```

```
        body { font-family: Arial, sans-serif; margin: 0; padding: 0;
background-color: #f4f4f9; }
        .container { max-width: 600px; margin: 50px auto; padding:
20px; background-color: white; border-radius: 8px; box-shadow: 0 2px
4px rgba(0, 0, 0, 0.1); }
        h1 { text-align: center; color: #333; }
        form { display: flex; flex-direction: column; gap: 10px; }
        input[type="text"] { padding: 10px; font-size: 16px; border:
1px solid #ddd; border-radius: 4px; }
        button { padding: 10px; font-size: 16px; background-color:
#5cb85c; color: white; border: none; border-radius: 4px; cursor:
pointer; }
        button:hover { background-color: #4cae4c; }
        .response { margin-top: 20px; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Customer Support Bot</h1>
        <form method="POST">
            <input type="text" name="user_query" placeholder="Ask me
anything..." required>
            <button type="submit">Submit</button>
        </form>
        {% if bot_response %}
        <div class="response">
            <strong>User:</strong> {{ user_query }} <br>
            <strong>Bot:</strong> {{ bot_response }}
        </div>
        {% endif %}
    </div>
</body>
</html>
```

- The HTML form allows users to input their query, which gets processed by the Flask app.
- The result of the query (user's input and bot's response) will be dynamically displayed using Jinja templating in Flask.

Step 5: Run the Application

You're ready to run your Flask app! Open your terminal and run the Python script:

```
python app.py
```

Visit `http://127.0.0.1:5000/` in your browser. You should see your chatbot interface. Enter a question, and the bot will respond based on the OpenAI GPT-3.5/4 model.

Step 6: Test and Deploy

Once you've tested your bot locally, you can deploy it to a cloud platform such as Heroku, AWS, or Google Cloud for broader access. Here's how you might deploy to Heroku:

1. Install Heroku CLI: `https://devcenter.heroku.com/articles/heroku-cli`

Create a `Procfile` in the root of your directory with the following content:

```
web: python app.py
```

2. Commit your project to a GitHub repository.

Push to Heroku using these commands:

```
heroku create
git push heroku master
```

Your customer support bot is now accessible to anyone via the web!

Additional Features

To make your customer support bot more robust, consider adding:

- **Multi-turn conversations**: Keep track of conversation history to provide more contextually aware responses.
- **Sentiment Analysis**: Use a sentiment analysis API to detect user emotions and adjust responses accordingly.
- **Knowledge Base Integration**: Integrate a database or CRM to provide personalized or data-driven responses.

Conclusion

By following this tutorial, you now have a working customer support bot using OpenAI's GPT-3.5/4 API. While this is a basic implementation, you can expand its functionality by adding

more sophisticated features such as real-time escalation to a human agent, multi-turn conversations, and integration with databases. This approach can enhance customer engagement while reducing the load on human support teams.

Let me know if you'd like to explore advanced features or need help with deployment!